TYPES OF MACHINE LEARNING

1. Supervised Learning

Regression

- Linear Regression
- Polynomial Regression
- Ridge & Lasso Regression
- Decision Tree Regression
- Random Forest Regression
- Support Vector Regression (SVR)
- Gradient Boosting (XGBoost, LightGBM, CatBoost)

Classification

- Logistic Regression
- Decision Trees
- Random Forests
- Support Vector Machines (SVM)
- k-Nearest Neighbors (k-NN)
- Naïve Bayes
- Gradient Boosting (XGBoost, LightGBM)
- Neural Networks / Deep Learning

2. Unsupervised Learning

Clustering

- k-Means Clustering
- Hierarchical Clustering
- DBSCAN (Density-Based)
- Gaussian Mixture Models (GMM)
- Mean Shift

Dimensionality Reduction

- Principal Component Analysis (PCA)
- Singular Value Decomposition (SVD)
- t-SNE
- Autoencoders

3. Reinforcement Learning

Key Methods:

- Q-Learning
- Deep Q-Networks (DQN)
- SARSA (State-Action-Reward-State-Action)
- Policy Gradient Methods
- Actor-Critic Methods
- Proximal Policy Optimization (PPO)
- Trust Region Policy Optimization (TRPO)
- Monte Carlo Methods

Data Preprocessing

Handling Missing Values

- df.isnull().sum() → Check the number of missing values in each column.
- df.dropna() → Remove rows or columns with missing values.
- df.fillna(value) → Fill missing values with a specific value (e.g., mean, median, or mode).
- df.interpolate() → Fill missing values using interpolation.

Feature Scaling

- StandardScaler() → Standardization (scales features to mean=0, std=1).
- MinMaxScaler() → Normalization (scales features to range [0,1]).
- RobustScaler() → Scaling using median and IQR (less sensitive to outliers).

Encoding Categorical Data

- LabelEncoder() → Convert categorical labels (e.g., "yes", "no") into numeric codes.
- OneHotEncoder() → Convert categorical features into binary dummy variables.
- OrdinalEncoder() → Encode ordinal categorical variables (with an order, e.g., low < medium < high).

Train-Test Split

Purpose

- Split dataset into **training set** (to train the model) and **testing set** (to evaluate performance).
- Helps prevent **overfitting** and ensures the model generalizes well.

```
from sklearn.model_selection import train_test_split

# Splitting dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,      # 20% data for testing
    random_state=42,   # ensures reproducibility
    shuffle=True,       # shuffles data before splitting
    stratify=y          # maintains class distribution (for classification)
)
```

Key Parameters

- test_size → Proportion of data for testing (e.g., 0.2 = 20%).
- train_size → Alternative to test_size, explicitly sets training proportion.
- random_state → Controls shuffling for reproducibility.
- shuffle → Shuffles dataset before splitting (default=True).
- stratify → Ensures balanced class distribution in classification problems.

Regression Models

Common Regression Models

- **Linear Regression** → Fits a straight line relationship.
- **Polynomial Regression** → Fits curves by adding polynomial terms.
- **Ridge Regression** → Linear regression with L2 regularization (controls overfitting).
- **Lasso Regression** → Linear regression with L1 regularization (feature selection).
- **ElasticNet Regression** → Combination of Lasso and Ridge.
- **Decision Tree Regression** → Tree-based non-linear regression.
- **Random Forest Regression** → Ensemble of decision trees, reduces variance.
- **Support Vector Regression (SVR)** → Uses hyperplanes for regression tasks.
- **Gradient Boosting (XGBoost, LightGBM, CatBoost)** → Powerful boosting-based regressors.

Classification Models

- Logistic Regression → Simple and interpretable for binary classification.
- K-Nearest Neighbors (KNN) → Classifies based on the majority vote of neighbors.
- Decision Trees → Splits data based on feature thresholds.
- Random Forest → Ensemble of decision trees, reduces overfitting.
- Support Vector Machines (SVM) → Finds hyperplane that best separates classes.
- Naive Bayes → Probabilistic model, great for text classification.
- Neural Networks (MLP – Multi-Layer Perceptron) → Used for complex non-linear decision boundaries.
- Gradient Boosting Models (XGBoost, LightGBM, CatBoost) → Advanced ensemble classifiers with high accuracy.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Create and train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate performance
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Evaluation Metrics

Regression Metrics

MAE (Mean Absolute Error):

- Average of absolute differences between predicted and actual values.

$$MAE = \frac{1}{n} \sum |y_{true} - y_{pred}|$$

MSE (Mean Squared Error):

- Average of squared errors. Penalizes large deviations.

$$MSE = \frac{1}{n} \sum (y_{true} - y_{pred})^2$$

RMSE (Root Mean Squared Error):

- Square root of MSE. Same units as target variable.

R^2 Score (Coefficient of Determination):

- Measures variance explained by model (1 = perfect fit, 0 = no fit).

Adjusted R^2:

- Modified R^2 that penalizes adding irrelevant features.

Classification Metrics

Accuracy:

- Ratio of correctly predicted instances.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision (Positive Predictive Value):

- Out of predicted positives, how many are correct.

$$Precision = \frac{TP}{TP + FP}$$

Recall (Sensitivity / True Positive Rate):

- Out of actual positives, how many were predicted correctly.

$$Recall = \frac{TP}{TP + FN}$$

ROC Curve (Receiver Operating Characteristic):

- Plots TPR (Recall) vs FPR (False Positive Rate).

AUC (Area Under Curve):

- Value between 0 and 1. Higher = better classifier.

Confusion Matrix:

- Tabular summary of TP, FP, TN, FN for detailed error analysis.

Feature Selection & Engineering

1. Feature Importance

- Use tree-based models like Random Forest, XGBoost, or LightGBM to rank features by importance.
- Helps identify which variables contribute the most to predictions.

2. PCA (Principal Component Analysis)

- A dimensionality reduction technique that projects data onto fewer dimensions while retaining maximum variance.
- Useful for datasets with many correlated features.

```
from sklearn.decomposition import PCA

# Reduce to 2 principal components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

3. Correlation Matrix

- Detects highly correlated features.
- Helps remove redundancy to avoid multicollinearity in models like Linear Regression.

```
import seaborn as sns
import matplotlib.pyplot as plt

corr = X.corr()
sns.heatmap(corr, annot=True, cmap="coolwarm")
plt.show()
```

Hyperparameter Tuning

1. GridSearchCV

- Exhaustively tests all possible combinations of hyperparameters.
- More accurate but can be computationally expensive.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define hyperparameters to test
params = {
   'n_estimators': [50, 100, 150],
   'max_depth': [None, 5, 10],
   'min_samples_split': [2, 5]
}

grid = GridSearchCV(RandomForestClassifier(), params, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best Score:", grid.best_score_)
```

2. Advanced Tuning Methods

- Bayesian Optimization (e.g., Optuna, Hyperopt): Uses probability to find optimal params.
- Genetic Algorithms: Inspired by evolution to optimize hyperparameters.
- Automated ML (AutoML): End-to-end optimization (e.g., Auto-sklearn, TPOT).

Clustering Algorithms

Popular Clustering Algorithms

- **K-Means Clustering** → Partitions data into k clusters based on centroid similarity.
- **Hierarchical Clustering** → Builds a hierarchy of clusters (agglomerative/divisive).
- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** → Groups points closely packed together while marking outliers.

```
from sklearn.cluster import KMeans

# Initialize K-Means with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)

# Fit the model on dataset X
kmeans.fit(X)

# Get cluster labels for each data point
labels = kmeans.labels_

# Optional: Get cluster centroids
centroids = kmeans.cluster_centers_
```

Deep Learning Fundamentals

Artificial Neural Networks (ANNs): Basic building blocks of deep learning.
Convolutional Neural Networks (CNNs): Specialized for image and spatial data processing.
Recurrent Neural Networks (RNNs): Designed for sequential data such as text or time series.
Transformers (BERT, GPT): Advanced models for natural language processing and large-scale learning.

```
import tensorflow as tf

# Build a simple feedforward neural network
model = tf.keras.Sequential([
   tf.keras.layers.Dense(128, activation='relu'),  # Hidden
layer with ReLU
   tf.keras.layers.Dense(1, activation='sigmoid')  # Output
layer for binary classification
])

# Compile the model
model.compile(
   optimizer='adam',
   loss='binary_crossentropy',
   metrics=['accuracy']
)
```